REMARKS

Applicants respectfully request that the above-identified application be reexamined.

The Office Action mailed on November 24, 2003 ("Office Action"), rejected all the claims in the application. Claims 1, 3, 4, 7, 9, 13, 15, and 16 were rejected under 35 U.S.C. § 103(a) as being unpatentable over Straub (U.S. Patent No. 5,430,878) in view of Nowlin, Jr. et al. (U.S. Patent No. 6,484,309). Claims 2, 5, 6, 8, 10, 11, 12, 14, 17, and 18 were rejected under 35 U.S.C. § 103(a) as being unpatentable over Straub in view of Nowlin, Jr. et al., as applied to Claims 1, 7, 13, and 15, and in view of Preisler et al. (U.S. Patent No. 5,675,803).

This amendment makes minor changes to the claims directed to eliminating typographical errors. Applicants submit that none of the amendments affect the scope of the amended claims.

For the reasons hereinafter set forth, applicants respectfully submit that the rejection of Claims 1-18, in view of the teachings of the cited references noted above, is in error and should be withdrawn and this application be allowed.

Prior to discussing the reasons why applicants believe that the claims of this application are clearly allowable, a brief discussion of the present invention, followed by a brief discussion of the cited and applied references, are presented. The following discussions of applicants' invention and the cited and applied references are not provided to define the scope or interpretation of any of the claims of this application. Instead, these discussions are provided to help the United States Patent and Trademark Office better appreciate important claim distinctions discussed thereafter.

Summary of the Invention

The present invention is directed to providing a method, computer-readable medium having computer-executable instructions, and system for patching computer application programs that are not compatible with the computer operating system executing the computer

LAW OFFICES OF
CHRISTENSEN O'CONNOR JOHNSON KINDNESS^PLLC
1420 Fifth Avenue
Suite 2800
Seattle, Washington 98101
206.682.8100

-8-

application program. More specifically, the method, the computer-readable medium having computer-executable instructions, and the system determine whether or not the computer application program is compatible with the computer operating system executing the computer application program. If the computer application program is determined to be incompatible with the computer operating system, a debugger is started to run the computer application program.

In one exemplary embodiment of the invention, a user starts an application. A determination is made whether the application is compatible or incompatible with the operating system. If compatible with the operating system, the computer application is run. If incompatible with the operating system, a debugger is started.

Application compatibility with the operating system in one embodiment of the invention is determined by checking application identity information against a database containing a list of applications that are currently known to be incompatible with the operating system. If the computer application program is found in the database, a set of attributes is checked to determine if the particular version of the application is incompatible. If the checked attributes match the ones in the database, the application is determined to be incompatible with the operating system. If the application is not found in the database, the application is determined to be compatible with the operating system. Based on the determination, the application is either run with or without a debugger. Applications that work correctly under the operating system are not required to go through an additional level of execution created by the debugger application. Thus, such applications execute faster.

In one embodiment of the invention, if an application is found to be incompatible with the operating system, the operating system starts a debugger application that, in turn, runs the incompatible application. Before loading the incompatible application, the debugger loads a dynamic link library (DLL) that contains patches for the incompatible functions of the application. Specifically, the DLL contains a list of breakpoints specifying the location where

-9-

the application needs to be patched, together with the appropriate code required to patch the application.

In this exemplary embodiment of the invention, when the debugger loads the DLL, a function is called that sets the breakpoints within the incompatible application. A breakpoint may be specified by (1) the module name and function name; (2) the module name and a memory offset; or (3) an address. The debugger also implements a handler that is capable of modifying the application code that is incompatible. In one embodiment of the invention, the debugger handler is cable of modifying the application code that is incompatible. In one embodiment of the invention, the debugger handler is capable of reading and writing memory within the incompatible application. Therefore, the handler is capable of modifying the incompatible code or inserting code into the application to fix the problem code contained within the application. For example, when the debugger reaches a breakpoint within the application, the debugger may be called to merely skip a portion of the incompatible code, or the handler may rewrite a portion of the code to contain a certain value in order to make the application compatible with the operating system.

One of the benefits of the use of a debugger application to patch incompatible applications is that this arrangement is very robust. The debugger is capable of monitoring every step an application takes while it is executing. Because the amount of code required to patch an application is generally very small, such as 200 bytes, patches are readily accessible to a user either through a Web site or FTP site.

Summary of the Cited and Applied References

Straub et al. (U.S. Patent No. 5,430,878) (hereafter "Straub")

Straub describes a method of examining and revising an image of a computer program so that the program can operate properly with the configuration of a computer. The method reads a computer application from a fixed storage medium. It then examines the program image to find

MSFT\14614AM2.DOC

whether the computer program recognizes the configuration of the computer it will run on. If it does not, the program image is revised so that it will recognize the configuration of the computer (first revision). If the computer program does recognize the configuration of the computer, then the method determines whether the program image can be operated from the lower memory area of the computer. If the program image cannot be so operated, the program image is revised so it can be run substantially anywhere in the computer's memory (second revision). If the program image can be run from the lower memory area of the computer, the computer operating system will execute the program.

The first revision is done by the computer checking a table maintained by the computer system. The table contains compatible computer program names and version numbers of operating systems that can run on the computer system. The computer references the table to obtain the operating system version number associated with the name of the computer program and stores this version number in the program image of the computer program.

The second revision is done by the computer replacing a predetermined set of instructions and data in the program image with another alternate set of instructions and data. This replacement enables the program image to be run from almost anywhere in the computer memory.

In summary, Straub merely provides a method for revising a computer program image in order for the computer program to be run on a particular computer configuration and anywhere in the computer memory. Straub teaches only two particular revisions—(1) storing operating system version number; and (2) replacing a predetermined set of instructions and data in the program image. Straub also teaches maintaining a table of compatible program names in the computer system. Straub does not teach or even remotely suggest using a debugger to execute the computer program. Nowhere does Straub teach or even remotely suggest using a DLL for

patching. Nor does Straub teach or even remotely suggest using breakpoint handlers with instructions for patching a program at debugger breakpoints.

Nowlin Jr. et al. (U.S. Patent No. 6,484,309 B2) (hereafter "Nowlin")

Nowlin purportedly teaches a method that converts a computer program designed for one operating system to run on a second operating system without a recompilation. This invention is specifically geared toward converting a non-Windows CE application program into a Windows™ CE-compatible computer program.

The method allegedly taught by Nowlin includes (a) loading a computer program on a computer system having a non-Windows CE operating system; (b) translating the computer program to run on another computer system having a Windows™ CE operating system; and (c) after the translation is completed, transferring the converted computer program to the second computer system running a Windows™ CE operating system.

The translation is accomplished by creating a translation layer on the first computer system. The translation layer communicates with a non-Windows CE system using a non-Windows CE system calling convention, and with a Windows™ CE system using a Windows™ CE system calling convention. The translation layer converts calls for a non-Windows CE operating system to calls that are compatible with a Windows™ CE operating system. The translation includes parsing executable and dynamic link libraries and using separate filters to translate each of these different file types. The translation includes modifying the header of a file so that a Windows™ CE operating system recognizes a file as a valid Windows™ CE file.

In summary, Nowlin merely provides a mechanism of parsing a non-Windows CE-compatible computer application and translating the necessary program code on the non-Windows CE computer system so that the application can be run on a Windows™ CE computer system without recompilation. Nowlin does not teach using a debugger to execute the application. Nor does Nowlin teach a DLL that contains a list of breakpoints. Neither does

-12-

Nowlin teach using handlers with instructions for patching a computer program at debugger breakpoints.

Preisler et al. (U.S. Patent No. 5,675,803) (hereafter "Preisler")

Preisler describes a runtime debug operation designated "Fix and Continue" operation. Preisler permits a user to begin a debugging session if an error is encountered in executable code. The user edits the corresponding source code to correct the error and then executes a "Fix and Continue" command without leaving the debugging session. The "Fix and Continue" command calls the compiler to recompile a source code file with the newly edited text, receives the resulting recompiled object code file from the compiler, uses a dynamic linker to link the recompiled object code into the target application program process, patches the previous versions of the same object code file to refer to the newly compiled code, resets any required variables and registers, and resets the program counter to the line of code being executed to where the error was discovered. The debugger then continues the debug session, thereby saving the time it would ordinarily take to quit the debug session, relink, and reload the target program, and start the debug session again.

Preisler's "Fix and Continue" command merely provides for pausing the debugging of a program to allow a programmer to add source code without leaving the debugger. Preisler nowhere teaches or even remotely suggests adding patches to a program based on the determinations that an application is incompatible with an operating system. Nor does Preisler teach or even remotely suggest breakpoint handlers with instructions for patching a program at debugger breakpoints.

The Claims Distinguished

A.   Independent Claims 1, 7, and 13

The Office Action has failed to show, and applicants have been unable to find, where any of the cited and applied references teaches or even remotely suggests the subject matter of

MSFT\14614AM2.DOC

independent Claims 1, 7, and 13, the only independent claims in this application. Claim 1 is a method claim, Claim 7 is a computer-readable medium claim, and Claim 13 is a system claim. More specifically, Claim 1 is directed to a method for patching a computer application program including a plurality of executable steps; Claim 7 is directed to a computer-readable medium having computer-executable instructions for patching a computer application program including a plurality of executable steps; and Claim 13 is directed at a computer system for patching a computer application program wherein the computer system is capable of running an application having a plurality of executable steps. Claims 1, 7, and 13 all recite:

> determining whether or not the computer application program is compatible with a computer operating system executing the computer application program; and
>
> if the computer application program is determined to be incompatible with the computer operating system, starting a debugger to run the computer application program

This subject matter is not taught or even remotely suggested by either Straub or Nowlin. Neither teaches starting a debugger to run a computer application program in order to determine if the computer application program is or is not compatible with the computer operating system. The Office Action alleges that Nowlin "suggested invoking calls to a debugger process." Applicants respectfully disagree. Applicants have been unable to locate the pertinent subject matter in the portions of Nowlin (Col. 2-Col. 3, line 39; Figures 2 and 3; and Col. 4, line 34-Col. 5, line 42) referenced in the Office Action. In this portion of text, Nowlin teaches using a linker to deal with memory issues in the executables. Applicants submit that a teaching of using a linker to address memory issues in executable files would not suggest to one of ordinary skill in the art the use of a debugger to patch incompatibility issues between a computer application program and a computer operating system. Therefore, there is simply no teaching or suggestion in

-14-

MSFT\14614AM2.DOC

Straub or Nowlin of the subject matter recited in Claims 1, 7, or 13. Thus, applicants submit that Claims 1, 7, and 13 are clearly allowable.

Since all of the other claims remaining in this application depend from Claims 1, 7, and 13, respectively, these claims are submitted to be allowable for at least the same reasons that Claims 1, 7, and 13 are allowable. Further, these claims are submitted to be allowable for additional reasons.

B.   Dependent Claims 2, 8, 14

Claims 2, 8, and 14 depend from Claims 1, 7, and 13, respectively. Each of Claims 2, 8, and 14 recites that the debugger is capable of running the incompatible application by:

(a)   setting at least one breakpoint within the application indicating a
        stopping point for the debugger;
(b)   running the steps of the application through the debugger;
(c)   monitoring the steps of the application as the application is running
        through the debugger to determine if the at least one breakpoint
        has been reached;
(d)   patching the application when a breakpoint has been reached; and
(e)   executing steps (b), (c), and (d) until the application has finished
        running.

As noted above, Claims 2, 8, and 14 were rejected under 35 U.S.C. § 103(a) as being unpatentable over Straub in view of Nowlin as applied to Claims 1, 7, and 13, and in view of Preisler. Applicants respectfully disagree.

First, the Office Action alleges that Straub's teaching of interrupting a revision process to insert a code is equivalent to setting a breakpoint for patching. Applicants respectfully disagree. The June 6, 2003, Office Action states: "a breakpoint is set when a request to jump to an outside trap code is initiated." Straub, however, merely teaches code insertion, and has no mention of initiating a request to jump to an outside trap code. Further, nowhere does Straub teach using a debugger; nor does Straub teach setting breakpoints so that code can be patched during the process of executing an application.

MSFT\14614AM2.DOC

Second, the Office Action alleges that Nowlin teaches executing code reconversion in a debugger. Applicants respectfully disagree. Applicants have been unable to locate any pertinent subject matter in the portions of Nowlin (Col. 6, line 40, to Col. 7, line 34) referenced in the Office Action. In this portion of text, Nowlin teaches dealing with memory issues in executable files using a linker instead of a debugger, as the present invention claims.

Thirdly, the Office Action alleges that Preisler teaches setting and monitoring for breakpoints and applying the patching process upon such breakpoints. Applicants respectfully disagree. The present invention recites patching the application when a breakpoint has been reached. Even if a patch site is the equivalent of a breakpoint, as suggested by the Office Action and which the applicants categorically deny, in Preisler, patching does not occur whenever a patch site has been reached. Preisler teaches using a check command, or uncheck command, by a user to indicate whether or not a particular patch site is to be patched.

Even assuming for purposes of argument that Preisler teaches a "Fix and Continue" debugger that meets the recitations of Claims 2, 8, and 14—which applicants categorically deny—this teaching does not make up for the deficiency of Straub and Nowlin to teach the underlying subject matter recited in Claims 1, 7, and 13. Further, there is no teaching or suggestion in Straub, Nowlin, and Preisler, taken alone or in combination, why it would be obvious to combine the individual teachings of these references. More importantly, as noted above, even if these references were combinable—which applicants categorically deny—the resulting combination would not anticipate the subject matter of Claims 2, 8, and 14 when the subject matter of these claims is considered in combination with the subject matter of Claims 1, 7, and 13, the claims from which these claims depend. Consequently, applicants respectfully submit that Claims 2, 8, and 13 are allowable for reasons in addition to the reasons why Claims 1, 7, and 13 are allowable.

-16-

MSFT\14614AM2.DOC

C. Dependent Claims 3, 9, and 15

Claims 3, 9, and 15 are dependent upon Claims 1, 7, and 13, respectively. Each of Claims 3, 9, and 15 recites that determining if the application is compatible or incompatible with the operating system comprises (a) determining if at least the one identifying attribute of a plurality of identifying attributes of a computer application program matches at least one identifying attribute of a plurality of identifying attributes of incompatible applications; and (b) if at least one of the identifying attributes match, determining that the computer application program is incompatible, otherwise determining that the computer application program is compatible. The Office Action alleges that Straub teaches the subject matter of these claims. Applicants respectfully disagree. Straub teaches using a table consisting of the names of compatible computer programs and the version numbers of the operating systems installed in the personal computer. Straub teaches identifying attributes of a computer program against attributes of compatible computer programs of a computer system. Nowhere does Straub teach matching identifying attributes of a computer program against attributes of incompatible applications of an operating system. More importantly, as noted above, even if it is obvious for one of ordinary skill in the art to modify the teaching of Straub to the claimed limitations in Claims 3, 9, and 15—which applicants categorically deny—Straub's teaching still would not anticipate the subject matter of Claims 3, 9, and 15 when the subject matter of these claims is considered in combination with the subject matter of Claims 1, 7, and 13, the claims from which these claims depend. Consequently, applicants respectfully submit that Claims 3, 9, and 15 are allowable for reasons in addition to the reasons why Claims 1, 7, and 13 are allowable.

D. Dependent Claims 4, 10, and 16

Claims 4, 10, and 16 are dependent on Claims 3, 9, and 15, respectively. Each of Claims 4, 10, and 16 recites that determining if the application is compatible or incompatible with the operating system further comprises (a) storing identifying attributes of incompatible

MSFT\14614AM2.DOC

applications; and (b) retrieving at least one of the stored identifying attributes for determining if at least one identifying attribute of a plurality of identifying attributes of the computer application program matches at least one of the stored identifying attributes of the incompatible applications. The Office Action alleges that Straub teaches the subject matter disclosed by these claims. Applicants respectfully disagree. As noted above, what Straub teaches is to store identifying attributes of compatible computer programs, such as program names, and the version numbers of the compatible operating systems of the personal computer. Nowhere does Straub teach storing identifying attributes of incompatible applications. More importantly, as noted above, even if it is obvious for one of ordinary skill in the art to modify the teaching of Straub to the claimed limitations in Claims 4, 10, and 16, which applicants categorically deny, Straub's teaching still would not anticipate the subject matter of Claims 4, 10, and 16 when the subject matter of these claims is considered in combination with the subject matter of Claims 1, 3, 7, 9, 13, and 15, the claims from which these claims depend. As a result, applicants respectfully submit that these claims are allowable for reasons in addition to the reasons why the claims from which these claims depend are allowable.

E.   Dependent Claims 5, 11, and 17

Claims 5, 11, and 17 depend from Claims 2, 8, and 14, respectively. Each of Claims 5, 11, and 17 recites that setting at least one breakpoint within the application comprises (a) loading a debugger dynamic link library containing a list of breakpoints, each breakpoint having a handler having a set of instructions for patching the application; and (b) the debugger accessing the list of breakpoints from the debugger dynamic link library and setting the breakpoints within the application. The Office Action alleges that Claims 5, 11, and 17 are anticipated by the combined teaching of Straub, Nowlin, and Preisler. Applicants respectfully disagree.

The Office Action alleges that Straub and Preisler combined teach "executing a debugger containing a set or list of breakpoint ... the debugger setting a set breakpoints within the

application." (*See* Office Action, page 6.) Applicants respectfully disagree. First, Claims 5, 11, and 17 recite a debugger dynamic link library, not the debugger itself, containing a list of breakpoints. Second, Straub teaches inserting one set of instructions so that the program image may run from computer lower memory region. Nowhere does Straub teach using a dynamic link library containing a list of breakpoints, each breakpoint having a handler having a set of instructions for patching the application. Nor does Straub teach the use of a debugger setting the breakpoints within the application. Third, even assuming for purposes of argument that Preisler discloses executing a debugger containing a set of breakpoints, each breakpoint having a set of instructions for patching an application—which applicants categorically deny—this teaching does not teach the use of a dynamic link library. What Preisler does teach is to identify the patch sites and accumulating information regarding these patch sites in a table. Nowhere does Preisler mention the use of a dynamic link library containing breakpoints.

The Office Action further alleges that Nowlin teaches using a debugger. Applicants respectfully disagree. A filter code does not perform the functions of the debugger recited in Claims 5, 11, and 17, as suggested by the Office Action. A debugger is a program that can run on its own in a computer system. A DLL, such as the filter code taught in Nowlin, needs to be loaded by a program. In addition, nowhere does Nowlin teach a filter code loading a debugger dynamic linked library containing a list of breakpoints. Nor does Nowlin teach a filter code setting breakpoints within an application.

Furthermore, there is no teaching or suggestion in Straub, Nowlin, and Preisler, taken alone or in combination, why it would be obvious to combine the individual teachings of these references. More importantly, as noted above, even if these references were combinable—which applicants categorically deny—the resulting combination would not anticipate the subject matter of Claims 5, 11, and 17 when the subject matter of these claims is considered in combination with the subject matter of Claims 1, 2, 7, 8, 13, and 14, the claims from which these claims

-19-

depend. As a result, applicants respectfully submit that Claims 5, 11, and 17 are clearly allowable for reasons in addition to the reasons why the claims from which these claims depend are allowable.

F.   Dependent Claims 6, 12, and 18

Claims 6, 12, and 18 depend from Claims 5, 11, and 17, respectively. Each of Claims 6, 12, and 18 recites that patching the application when a breakpoint has been reached comprises (a) calling the handler associated with the breakpoint; and (b) patching the incompatible application based on the instructions within the handler. As pointed out above, Straub does not teach using breakpoints. Nor does Straub teach calling a handler associated with a breakpoint or patching the incompatible application based on the instructions within the handler. Further, as pointed out above, Preisler does not teach patching upon reaching a patch site. In Preisler, a user determines whether to patch or not at a particular patch site. Furthermore, as pointed out above, Nowlin does not teach the use of a debugger DLL, nor the use of breakpoints, or a handler associated with a breakpoint.

Therefore, the cited references, taken alone or in combination, do not teach or even remotely suggest the subject matter of Claims 6, 12, and 18. Further, there is no teaching or suggestion in Straub, Nowlin, and Preisler, taken alone or in combination, why it would be obvious to combine the individual teachings of these references. More importantly, as noted above, even if these references were combinable, which applicants categorically deny, the resulting combination would not anticipate the subject matter of Claims 6, 12, and 18 when the subject matter of these claims is considered in combination with the subject matter of Claims 1, 2, 5, 7, 8, 11, 13, 14, and 17, the claims from which these claims depend. As a result, applicants respectfully submit that Claims 6, 12, and 18 are allowable for reasons in addition to the reasons why the claims from which these claims depend are allowable.

MSFT\14614AM2.DOC

CONCLUSION

In view of the foregoing comments, applicants respectfully submit that all of the claims in this application are clearly allowable in view of the cited and applied references. Consequently, early and favorable action for allowing these claims and passing this application to issue is respectfully solicited.

Respectfully submitted,

CHRISTENSEN O'CONNOR
JOHNSON KINDNESS^{PLLC}

Joy Y. Xiang
Registration No. 55,747
Direct Dial No. 206.695.1607

I hereby certify that this correspondence is being deposited with the U.S. Postal Service in a sealed envelope as first class mail with postage thereon fully prepaid and addressed to the Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450, on the below date.

Date: March 1, 2004

JYX:mgp

MSFT\14614AM2.DOC